Persistency of Objects

WorkShop on Objects, XML, and Databases

OOPSLA 2001, Tampa, Florida

Jim Kowalkowski, Marc Paterno

1 Introduction

Fermilab is a national laboratory which engages in research into the fundamental nature of matter. It is the home of the world's highest energy accelerator, the Tevatron, which collides protons and anti-protons at an energy of two billion electron-volts (2 TeV). The Tevatron is the host to two general-purpose detectors, CDF and DØ. Each of these detectors consists of approximately a million channels (sensors). The detectors are exposed to proton-antiproton collisions at a rate of several millions per second; the data read from the detector in response to one of these collisions is called an event.

After several layers of filtering to remove uninteresting events, we end up with a 50 Hz stream of events of about 250 KB each. The events go through a series of processing stages we call reconstruction, which produce derived data that eventually represent the physics processes that were present in detector at the time the sensors were read. Several database technologies are used to store and manager these data. The DØ and CDF experiments are beginning to collect data that is expected to exceed two petabytes over the next three years.

An event reconstruction program processes a stream of events. This program is written entirely in C++. It consists of about 200 libraries and more than 200 persistent objects. For this program to be run, it needs a configuration scripts indicating the task it is to perform and the parameters specific to that task, detector state information (*i.e.* voltages, temperatures), sensor calibration constants, detector alignment constants, and a set of parameters describing the stream of events to be processed. Each of the concepts above (calibration, geometry, configuration, alignment, and detector state have objects associated with them. The event can be thought of as a container object that is composed of many objects, each representing a real thing found in an event (an electron, a discharge in a drift chamber, a light pulse in a phototube).

The term database is used in a broad sense within the reconstruction program. Information such as the calibrations, alignment, state, and configuration (auxiliary data) are direct access in nature and are stored in a RDBMS. Events are categorized and described in a RDBMS event catalog. The processing of events is essentially sequential in nature, but the file that holds a series of events can easily be viewed as a small database that allows navigation around the file, describing the characteristics and summarizing events within the file. Each event can also be viewed as a database of all the things present in the detector at the moment of the readout.

In the course of our work supporting several large experiments at Fermilab, we have used a variety of different distributed computing and database technologies. We have not been completely satisfied with any of the solutions we have used, and are thus searching for an improvement. In this paper, we sketch some of the solutions we have used so far, and describe the features of those solutions which we have found unsatisfying.

2 Position

We have arrived at the following observations because of the problems we have encountered while helping to design and implement the infrastructure of the reconstruction programs for CDF and DØ. They are strongly influenced by the environment and culture in which we work.

2.1 Concerning Objects

The optimal design for a specific object usually depends on the language in which the design is to be implemented, and the detailed design often depends on features of the particular language, for example templates in C++ or reflection in Java™. Several of the products we have used have put restrictions on the use of some language features, in order to gain interoperability between languages at the object level. We find this unacceptable.

If a system is going to gain the widely-touted benefits of object-oriented design, it must respect encapsulation. Specifically, it may not manipulate the data belonging to an object through anything other than the interface of that object.

Making an object usable from multiple languages is difficult -- especially so when some of the languages are ones with no support for object-oriented design, such as FORTRAN-77. Making data structures usable from multiple languages is less difficult. This causes an immediate tension with the point immediately above.

We have not had success in providing a language-neutral interface to objects, because of troubles in language features and performance. We have found using a lowest-common-denominator object definition to support multiple languages unsuitable.

2.2 Concerning Persistency

Our problem domain (the reconstruction of events) is data-centric. While there are clear benefits to be gained from an object-oriented design of the reconstruction software, we believe that an object-oriented view of the persistent data is not useful. Instead, we find that a set of object-oriented wrappers used to manipulate relatively simple data structures to be more successful.

Determining the proper point of division between information describing the objects in an event and the information in the objects themselves is difficult. We have not yet determined reliable guidelines for deciding when to make subobjects available as data (for example, as columns in a RDBMS) and when to treat the subobjects as opaque (for example, as a BLOB in a RDBMS), interpretable only through the object wrappers of the system.

Schema evolution, the process of managing changes of an object over time, is a source of trouble with real objects, because the data is bound to a version of the code. We find that separating the division of the system into data and object wrappers helps solve the problem.

In our culture, locking users into a single technology is often unacceptable. Locking users into a single product is still less acceptable, and a single commercial product is still less so. “Open standards” are very important, so proprietary interfaces and languages are frowned upon (Java™ is the notable exception here). Open source products are preferred, when available.

Intrusive persistency mechanisms (ones that require specific implementation in your classes, rather than requiring some interface or protocol) are burdensome. They often interfere with the natural implementation of classes, and limit what language features one may use. They also make testing much more difficult; the burden of recompiling a set of several hundred libraries to test a new feature of the persistency mechanism often means that such testing is not done, or is done to a much lesser extent. This also makes the exploration of new design options much more costly. We strongly prefer non-intrusive mechanisms.

The code that translates a persistent representation to the object representation can consume large amounts of memory and CPU time. This can grow to dominate some simple (but important) data processing steps, and so it is important to make this as small and fast as possible.

2.3 Concerning Database Technology

We have not seen any object-oriented database that will meet our needs.

We have seen RDBMSs that work for the auxiliary data (configuration, geometry, calibration, and event catalog information). The access patterns for these data are suited to the tools provided by RDBMSs, and are reasonably predictable. The access times are acceptable.

HEP community-developed solutions are in use for the handling of the event data. No other solutions have been demonstrated to be sufficient for handling the data volumes and access patterns.

3 Future Directions

3.1 Summary of the Past

Here we present a list of some of the features of the software systems we are involved in. In each case, there are advantages. The purpose of this section is to focus on some of the problems we are currently encountering as the detectors are coming online.

3.1.1 Event Data Objects

The current set of data objects inside the event are BLOBs, essentially unusable without the C++ object. The persistent form of objects within the event is very C++ oriented. The data objects are streamed in from buffers and out to buffers by methods in the object or by a data dictionary that has intimate knowledge of the memory layout of the object.

In many cases, it is convenient for portions of the data objects to be analyzed in a spreadsheet or relational table view. The BLOBs make this nearly impossible. Maintaining streaming code within the object is difficult when it comes to versioning and difficult to migrate to a format that is not a BLOB. Producing a data dictionary management system for C++ is a large project that involves parsing C++ header files, and in our case it has a list of restrictions and rules associated with objects that it can work with.

3.1.2 Database Access

The system has applications connecting directly to the database and applications connecting through an implementation of a three-tier architecture. There is also support for multiple database accessors built into a single application (similar the ODBC data source concept). Code generators are used in one of the projects to produce get and put routines to retrieve and store data. The use of code generators greatly simplifies the task of adding information to the database and making it readily available in C++.

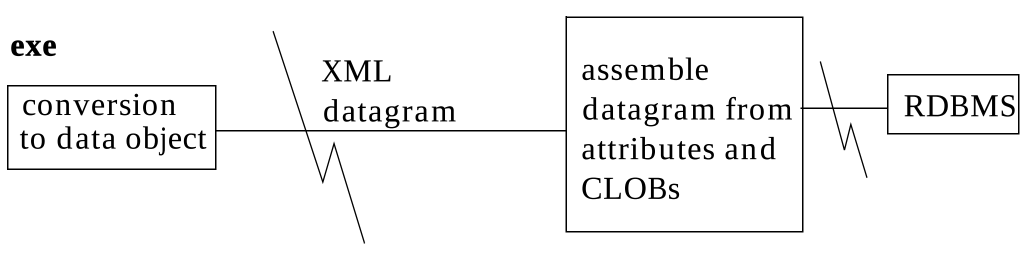
Synchronizing code and data structures between the client application and the server that connects to the database is a maintenance burden and difficult to test. The client applications cannot change data formats without the server changing. Supporting multiple databases within one application is a maintenance burden; the code to read and translate the database information into objects needs to synchronized at all times. Supporting multiple vendor-supplied protocols or client libraries within the applications is difficult. Accessing the databases from C++ that we use requires understanding a complex APIs, which means there is a substantial support and maintenance burden associated this the code.

3.2 Auxiliary Data

We are currently researching the use of HTTP as the protocol for the transfer of auxiliary data to and from reconstruction programs. We plan to continue our use of RDBMSs, and to communicate with the RDBMSs through one of the widely supported “standards” for various languages (JDBC, ODBC, Python DBI, Perl DBI, etc.), via a servlet or its equivalent. We plan for the data in the messages to be an XML representation of the data portion of the object being transferred. We see the following advantages of this arrangement:

- It uses a well established, mature, and relatively simple protocol.
- Clients implementing the protocol are freely available for all the languages in which we have interest.
- The database access is achieved through well-established methods.
- We hope that XML will be able to support the management of changes in object format, by including sufficient metadata.
- It allows us to provide a single point of support for code that transforms information for each database.
- Neither clients nor servers are not tied to a particular RDBMS.
- Servers can use any language and any available API for a specific database, while clients of that server need to know nothing about which language or API is used. Changes in the choices made for a server do not affect any clients

Clients can be implemented in any language, independent of what was chosen for the server. Different clients can be written in different languages.



We are still trying to understand when to store the information as CLOBs or BLOBs and when to break it up into attributes within tables. Much of these decisions will be based on how well the system performs and how the information will be analyzed (column wise for example). The prototype system will help determine the overhead of adding the XML tags to the data, the overhead of the conversion to object format, and the differences between a text representation and a binary one. It will also help us to understand how to separate the data - attributes versus database CLOBs. We will also be evaluating how well suited this technology is for our environment.

3.3 Event Data Management

We are still working through ways to manage the data held with the events and within the files. We are hoping that this workshop will help lead us to a better way to manage this information. The files are about 2 GB in size. It is important to have summary, indexing and property information about the events in the file so the application can quickly determine if the event in the file are interesting enough to be processed. Here are some of the questions we are trying to answer:

- Is it practical to store events in something other than the file format we have now?
- Does it make sense to use XML as the persistent form for many of the data objects held within the event? Can the event summary information be made available in XML?
- Can XML be used in the context of our current file format? Will it help with language interoperability issues and object versioning?